Comparative Study of Computer Languages

Input Output Statements(3)

Part XII

R. Ramaswamy

The input statements give instructions to get the data into the program and the output statements give instructions to output the result on paper.

As in the case of FORTRAN and COBOL we have three methods for getting the data into the program in BASIC. They are:

1. Entering the data through the console.
2. Entering the data as a part of the program in the program itself.
3. Entering the data by asking the computer to read a data file kept separately.

We shall study the first two methods in this article and consider the third method later when we study about files.

Input statements in BASIC

The name of the statement which gives instructions to get the data through the console is called the INPUT statement. The general form of the INPUT statement is

```
INPUT list
```

The word 'INPUT is a code word which means that the computer is asked to get the data through the console. The list contains the variables separated by commas, whose values are entered or keyed through the console by the operator. Suppose one writes an INPUT statement as

```
INPUT A, B, C
```

On encountering the INPUT statement, the computer stops and puts a ? mark. Then the operator must key in the values of the variables one by one, separating each value by a comma. After entering three values corresponding to the three variables in the list, the computer proceeds to process the other statements in the program. Of course, the type and the number of variables in the list and the type and the number of values keyed through the console must match, otherwise an error condition will result.

When the computer simply puts a ? mark, the operator must know how many values he has to key. So we can give a prompt string to the operator to know how many values he has to key through the console by adding the prompt string within quote marks as shown below.

```
INPUT "ENTER THE VALUES OF A,B,C,", A, B, C
```

When the computer encounters the above INPUT statement, it simply prints the words within the quote marks without putting a question mark. Obviously this string is meaningful to the operator and so he starts keying the values of the variables A, B and C as displayed in the prompt string. The computer will proceed further only after getting the values of A, B and C.

Interactive mode or conversational mode

The above mode of entering data into the program through the console is called the interactive mode or the conversational mode. In this mode you can establish a dialogue between yourself and the computer. The computer will ask for the data by displaying the prompt string on the screen. You enter the data by keying on the console. If you enter wrongly, the computer gives a message on the screen as

"REENTER". The operator then re-enters the data. Thus the conversation can continue. Though this conversational mode is possible in other languages, it is found that BASIC is most suitable.

Statements for entering data in the program itself

The statement or the instruction to enter the data in the program itself is called the READ-DATA statement. The two key words READ and DATA go together, although they do not appear in the same statement or necessarily follow each other directly. However, for every variable listed after the word READ in a statement, there must follow a statement with the word DATA at some point in the program followed by the constants.

The computer matches the first READ variable with the first DATA constant, the second variable with the second DATA constant and so on. The DATA statement can be placed anywhere in the program as long as it precedes the END statement. The following example illustrates the READ-DATA statement.

```
READ A, B, C, D$
..
..
DATA 27.6, 789, 0, "KRIS"
```

The variables in the READ list and the constants in the DATA list are separated by commas. The data can be either in the integer mode or in the real mode, the only restriction being that the numeric data must correspond to the numeric variables and the string data must correspond to the string variables. We can also write the above READ-DATA statement as follows:

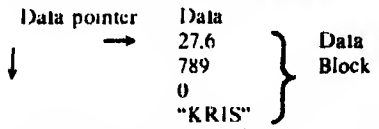
```
READ A, B, C, D$
..
..
DATA 27.6
DATA 789, 0
DATA "KRIS"
```

We observe that the number of DATA statements need not be equal to the number of READ statements, but the number of constants must be equal to the number of variables appearing in the READ list. If the number of variables exceeds the number of constants, the computer will give an 'out of data' error message. If the number of constants exceeds the number of variables, the surplus constants will be ignored.

Since READ statements generally precede the DATA statements, the usual practice is to place the READ statements at the beginning of the program and the DATA statements at the end before the END statement.

Visualising the data access in the READ-DATA statements

One can visualise the method of data access by the BASIC compiler in the READ-DATA statement. The BASIC compiler organises the contents of all the DATA statements in the program into a data block. During program execution one can visualise a pointer to move down the data block. When the computer reads the value of A, the pointer is directed to the data 27.6, somewhat as shown below. As soon as it reads the first data, the pointer moves down and positions itself against the next data and so on.



Sometimes it may be required to read the same set of data in another set of variables, say, E, F, G, H$. In that case the pointer can be restored to its original position by making use of a statement called the RESTORE statement.

RESTORE statement. The RESTORE statement transfers the pointer, regardless of where it is, to the first value of the data block. Suppose one writes

```
READ A, B, C, D$
DATA 27.6, 789, 0, "KRIS"
RESTORE
READ E, F, G, H$
```

As soon as the first READ statement is executed, the pointer comes down below the last data. At that time it is not pointing towards any data. As soon as the computer encounters the RESTORE statement, the pointer moves up and positions itself against the first data. So when the second READ statement is encountered, the same set of four values are read into the variables E, F, G, H$. If the RESTORE statement is not there, the computer will give an 'out of data' error message. Any number of RESTORE statements can be given. Everytime the computer encounters the RESTORE statement, the pointer is simply returned to the top of the data block.

The READ-DATA statement in BASIC is analogous to the DATA statement in FORTRAN and the VALUE clause statement described in the WORKING STORAGE SECTION in COBOL. In all these cases, the data is entered in the program itself. In BASIC, no formatting is required, i.e. the descriptions about the type of data, the length of data, etc are not required as in the case of FORTRAN or COBOL.

Output statements for printing the result on paper

The PRINT statement is used to output numerical or string data in the printed form. The general form of the PRINT statement is

```
PRINT list
```

The word PRINT is a code word to tell the computer to print the values. The list contains the list of variables whose values are to be printed. The variables are separated by commas. Suppose one writes

```
PRINT A, B, C, D
```

The values of A, B, C and D are printed in one line according to the built-in format.

Normally the printer length is 80 columns. This length is divided into four zones of 20 columns each. Each value is printed starting from the first column of each zone. If the values of A, B, C and D are 23.5, 45, 67.8 and 80 respectively, they will be printed somewhat as shown below:

1st zone	2nd zone	3rd zone	4th zone
23.5	45	67.8	80

You may notice that the first column in each zone is blank. This space is for the sign. If the number is positive the first column will be left blank. If the number is negative the sign will be printed in the first column. If the list contains more than four variables, the items will be printed in the next line. So if the data items cannot be accommodated in one line, the printer will automatically go to the next line and print the data items zone-wise.

Suppose you want to print the data items closer, you can use semicolons as separators between successive variables. In the case of numbers one space will be left between two data items and in the case of strings no space will be left between the data items. Suppose one writes

```
PRINT A; B; C; D
```

Then the computer will print the values of A, B, C and D continuously, leaving one space between the successive values as follows:

```
23.5  45  67.8  80
```

Suppose the value of A$ is "SANKARA", the value of B$ is "SUBRA" and the value of C$ is "MANIAN", and if one writes

```
PRINT A$; B$, C$
```

the computer will print the result as

```
SANKARASUBRAMANIAN
```

The semicolons between string variables in the PRINT list will concatenate the string data items.

In case you want to print the result with some descriptive items, the descriptive items can be put within quotes and the PRINT statement written as follows:

```
PRINT "VOLUME OF THE CYLINDER IS", X
```

If the value of X is 4.58, the computer will print as

```
VOLUME OF THE CYLINDER IS 4.58
```

In the case of PRINT statements, which we have considered so far, the computer prints the result in a predefined format. It does not give the programmer sufficient facility to place the output data in the required locations. To overcome this difficulty, BASIC has another provision called the TAB function statement.

The TAB function

The general form of the TAB function is

```
TAB(X)
```

where X can be either a variable or an expression or a constant. Of course the variable, the expression and the constant must all be numeric. The expression X will be evaluated first and its integer value will be taken to position the printer before starting any printing operation. Suppose one writes

```
PRINT TAB(15); X
```

The TAB(15) function causes the value of X to be printed from the 15th column onwards. Once the TAB function is introduced, the permitted delimiter is the semicolon.

More than one TAB function can be used in the same PRINT statement to position more than one variable in the printer line. Suppose one writes

```
PRINT TAB(10); "RAJA"; TAB(14); "GOPALAN"
```

The above statement will generate the output from the 10th column onwards as

```
RAJAGOPALAN
```

Now, if one writes

```
PRINT TAB(10); "RAJA"; TAB(12); "GOPALAN"
```

We find that the value of the second TAB is less than the current print position and so the print value is shifted to the second line with the same TAB setting and the output will appear as

```
RAJA
  GOPALAN
```

It must be noted that everytime TAB function is executed in the same statement, the printer position is counted from the beginning of that line and not from the position it completed the printing of the previous value.

Print using statement

Sometimes it may be required to output only specified number of characters, either digits or strings in specified locations justified whether right or left. This can be done by representing the position of each digit or character by some special symbol called the image. For example, the image symbol for numeric digits is # (pronounced as hash).

Suppose you have a number 234.5679 stored in the location A and you want the computer to print the value rounded off to two digits to the right of the decimal point. One gives the specification as follows:

```
PRINT USING "####.##"; A
```

With the above specification, called the integer image specification, the value of A is printed only with two digits to the right of the decimal point, i.e., rounded off to two decimal places. The output will appear as

```
234.57
```

The remaining digits after the second decimal place are simply truncated. Suppose you have four values to be output with different number of digits with different spacings between them, the specifications can be written as

```
PRINT USING "####.## #### #####.# ##"; A, B, C, D
```

The hash symbol has got the necessary provision to suppress non-significant zeros.

Consider you have a number 00987.654 located in A, the specification for this will be given as

```
PRINT USING "#######.##"; A
```

The computer will print the result as

```
987.65
```

You can notice that the two zeros in the non-significant positions have been suppressed. Again notice that the result in a numeric field is always right justified and this is what we require for the sake of alignment of the columns.

The USING statement can also be given as a separate assignment statement and used later in any number of PRINT statements. For example, one can write

```
FOR$ = "####.## ##### ##.## ########"
PRINT USING FOR$; A; B; C; D
```

Just as we write the format statement in FORTRAN separ-

ately, in BASIC also the format statement can be separately written as shown above. The same format statement can be used for any number of PRINT statements if required.

String image specification

We have said that the positions of numeric digits are symbolically represented by the hash sign. The hash sign is called the integer image specification. In a similar way a fixed length string data field of more than one position is specified by a pair of slashes (/) separated by zero or more characters. The width of the field is equal to the number of characters between the two slashes plus two. Any character can be placed between the slashes. These characters will be ignored by the computer and it will take them only as symbols for printing string data.

The string will be printed left justified in the given field width. If the string is longer than the field width, then it will be truncated on the right. If the field is longer than the string, then the string will be printed from the left end of the field, leaving blanks on the right end, i.e. we say that the strings are printed left justified in the given field. Suppose we give a print statement as follows:

```
FOR$ = "/ - - - - - - - - - - - - - - - - - - - - /"
PRINT USING FOR$, "R. KRISHNAN"
```

The FOR$ field is 22 characters long. The string data is only 11 characters long. The name R. KRISHNAN will be printed in the first 11 positions and the remaining 11 positions on the right will be left blank. We can use a single ampersand symbol (&) to represent strings of different length. Suppose one writes as follows:

```
FOR$ = "&"
PRINT USING FOR$; A$
```

In this case the length of the string stored by the variable A$ can be anything within the permitted length for a string constant. The full string will be printed without any truncation by the above FOR$ specification.

Print statement without any list

To give vertical spacing, the PRINT statement without any list is used. One dummy PRINT statement will skip to the next line. Suppose you want to print two items A and B in successive lines, one has to write as follows:

```
PRINT A
PRINT
PRINT B
```

But if you want to skip two lines, you have to give two dummy PRINT statements.

The purpose of the format statements is to tell the computer where and how the result should be printed on the paper. With some experience one can write many complex printing formats by combining different specifications depending on the need. In BASIC there is no format statement for reading data. Only for printing purposes formats are required in BASIC. If no formats are given the result will still be printed, but in that case the computer will use the built-in format for printing purposes. In FORTRAN formats are required both for reading and printing the data. In COBOL also formats are required both for reading and printing the data.

In BASIC formats can be either included in the print statement or given separately. In FORTRAN and COBOL formats have to be given by separate statements. In FORTRAN and BASIC the formats statements can be placed anywhere in the program. But in COBOL the format statements can be placed only in the DATA DIVISION of the program. The way of coding the formats in these languages are different.

Input statements in PL/I

We have three ways of entering data into a PL/I program same as in the case of FORTRAN, COBOL and BASIC. These are:

1. Entering the data through the console.
2. Entering the data in the program itself.
3. Entering the data in the program by reading a file.

In this article we will consider the first two methods and study the third method later.

Input statements for entering the data through the console

The general form of the input instruction or the statement for entering the data through the console is

```
GET LIST(list)
```

The two words GET and LIST are key words which have special meaning to the computer. The word list inside the parentheses contains the list of variables whose values are to be entered through the console. The variables in the list are separated by commas. Suppose one writes

```
GET LIST (A, B, C)
```

The computer on encountering this statement will stop and wait until the operator keys in the data values for A, B and C. The variables can be either integers or reals or character strings or bit strings. This information must have already been given in the declaration statement in the beginning of the program. We have already said in the article on coding rules for variables, that every variable introduced in a PL/I program must be declared as belonging to any one of the four categories. The declaration statement itself will be used as the format for reading the input data. There is no separate format statement required in PL/I.

The values that are keyed in must conform to the specifications of the variables declared earlier, otherwise an error message will be given by the computer. The GET LIST statement in PL/I is analogous to the READ statement in FORTRAN, the INPUT statement in BASIC and the ACCEPT statement in COBOL. Though the above statements are intended for doing the same function of getting the data into the program through the console, the way of coding the instructions in the different languages are different.

Input statements for entering the data in the program

In PL/I, the data values for the different variables can be

initialised in the declaration statement itself. The data values can be either numeric or string. Suppose you want to initialise the values of three variables A, B and C then you write down the declaration statement as

```
DCL (A, B, C) FLOAT BINARY STATIC INIT (12.5, 345, 79);
```

The word STATIC is a code word which tells the computer that the variables having this attribute can have their values initialised. INIT is another code word which simply means initial values. The actual values of the variables A, B, and C are enclosed within parentheses immediately after the code words STATIC INIT. The first part of the declaration is the same we have seen earlier.

Suppose one wants to give initial values for strings, e.g., to give values for two variables M1 and M2 as JANUARY and FEBRUARY respectively. Then the statement is

```
DCL (M1, M2) CHAR(10) VARYING STATIC INIT ('JANUARY',
'FEBRUARY');
```

The string values must be enclosed within quotes. The other key words have the same meaning which we have said earlier. The DCL-STATIC-INIT statement in PL/I is analogous to the DATA statement in FORTRAN, the VALUE clause statement in COBOL, and the READ-DATA statement in BASIC.

Output statements to print the result on paper

Just as in BASIC, we have both formatted and unformatted print statements. The general form of the unformatted print statement in PL/I is as follows:

```
PUT LIST (A, B, C, D);
```

A, B, C and D can be either numeric or string. Of course, they must have been declared in the beginning of the program, as it is mandatory to declare every variable that is used in a PL/I program. If the variables are numeric, the values will be printed in the exponent notation with seven significant digits in the mantessa part. If the values of A, B, C and D are 1, 2, 3 and 4 respectively, then the computer will print the result as follows:

```
1.000000E+00 2.000000E+00 3.000000E+00 4.000000E+00
```

The computer divides the print line into zones, each zone having a width of 14 columns. The first value is printed in the first 14 columns, the second value is printed in the second 14 columns and so on. If there are more items, printing will be continued in the next line. The programmer has no control over the printing positions of the values. The computer is said to print the result according to a built-in format. Strings also will be output in a similar manner. Suppose you want to print the values of A and B in separate lines, you have to write as follows:

```
PUT SKIP LIST(A);
PUT LIST(B);
```

The key word SKIP will cause the movement of the print head to the next line after printing the value of A. The value of B will then be printed in the first zone of the next line. The PUT SKIP statement without any list will cause the movement of the print head to the next line without printing anything in that line. This statement is useful for giving vertical spacing between the output values.

Formats for output data items

Though no formats are required to read data items, it is necessary to have formats for presenting the output in a meaningful way. The format for each of the output data item is given in the output statement itself. The general form of the output statement with the format specifications for each data item is as follows:

```
PUT EDIT (list) (format list)
```

The code word EDIT tells the computer that the data items in the list must be printed as per the format list given at the end of the variable list. We will now introduce the format specifications.

A format. The general form is A(w), where A is a code word to tell the computer that the data item is a string and w represents the width of the string. If w is omitted, then w is assumed to be the length of the output string. If w is greater than the output string length, then blanks are added to the right. If w is less than the output string length, the string is truncated from the rightmost position.

F format. The general form is F(w,d), where F is a code word to tell the computer that the data item is numeric, w represents the width of the field or the number of digits in the field and d is the number of digits to the right of the decimal point. This format is used to output the data item in the floating point notation.

X format. This format is used to skip columns in a line. The general form is X(w), where X is a code word to tell the computer that it must move its print head by w columns in the same line.

Suppose you want to output four data items A, B, C and D whose values are 'KRIS', 23, 42.1 and 432.1 respectively. One can write the formats in the output statement as follows:

```
PUT EDIT (A, B, C, D) (A(4), X(2), F(3), X(2), F(4,1), X(2), F(5,1))
```

The printed output will appear as follows:

```
KRIS 23 42.1 432.1
```

Now if one wants to print the values of A and B in one line and the values of C and D in the third line, one writes the statement as follows:

```
PUT EDIT (A, B, C, D) (A(4), X(2), F(3), SKIP(2), F(4,1), X(2),
F(5,1))
```

The word SKIP(2) causes the print head to skip two lines, counting being done from the current line. In general, SKIP(n) skips n lines and gives n-1 blank lines.

REMOTE format. The format specifications can also be given as a separate statement with a label put before it. Then this label is referred to in the format list. The advantage is that this labelled format can be used with a number of output statements. This type of formatting is referred to as REMOTE FORMAT in PL/I.

We give some label to the format list alone and write:

```
L1: FORMAT(A(4),X(2),F(3),X(2),F(4,1),X(2),F(5,1));
```

L1 is called the label or name of the format statement. The name followed by the colon mark enables the computer to identify it as the label of the statement. In FORTRAN we

give numbers to statements for reference purposes. In PL/I we give names coined by alphabets or alphabets and numbers. A statement label is coined in the same way as we coin the name of a variable. To print the values of A, B, C and D as per the above format, we write

PUT EDIT (A, B, C, D) (R(L1));

The letter R is a code to indicate that the format is a REMOTE FORMAT and it is given in the statement which is labelled by the name enclosed within parentheses immediately following the letter R. The same format can be used by any number of output statements if required.

Formats can be given for printing reals in the scientific notation or the exponent notation. The general form is F(w,d), where w defines the field width and d gives the number of digits to the right of the decimal point. The output will appear as

bddddddEee

So the format which one must give is F(13,6). The total width of the item is 13. One space is left for the sign, one for the decimal point, one for the integer part of the number, six for the fractional part of the number and four for the exponent coding part. The F(13,6) format can output the biggest as well as the smallest number that can be output by the PL/I program. The minimum value of w is d+7. If w is less than d+7, an error message will be given. If w is greater than d+7, the number will be printed right justified in the field leaving blanks on the left.

(To be continued)